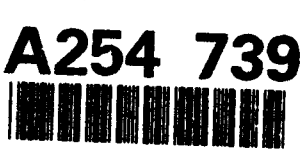

TECHNICAL REPORT BRL-TR-3382

BRL

HARDWARE AND SOFTWARE DESCRIPTION OF A PROTOTYPE CONTROLLER FOR THE TWO-DEGREE OF FREEDOM "BRL" MOUNT

MARK D. KREGEL

JULY 1992

U.S. ARMY LABORATORY COMMAND

BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute indorsement of any commercial product.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1992	Final, January–July 1989

4. TITLE AND SUBTITLE

Hardware and Software Description of a Prototype Controller for the Two-Degree of Freedom "BRL" Mount

5. FUNDING NUMBERS

WO: 44592-102-51-4233

6. AUTHOR(S)

Mark D. Kregel

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-5066

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

BRL-TR-3382

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

As part of the LABCOM TEAM COOP Program, the U.S. Army Ballistic Research Laboratory recently constructed a two-degree of freedom mount suitable for slewing loads up to 800 lbs. The mount was driven in both azimuth and in elevation by large stepper motors. In order to test and evaluate the mount, a preliminary controller was constructed that would allow its operation through keyboard entry of a remote computer. This report describes the controller and its operation as well as technical descriptions of key circuit elements. It also provides a listing of the controller software and the corresponding software for the remote computer.

14. SUBJECT TERMS

continuous processing; microprocessors; gun mounts; circuit analysis

15. NUMBER OF PAGES

59

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

INTENTIONALLY LEFT BLANK.

LIST OF FIGURES

INTENTIONALLY LEFT BLANK.

1. INTRODUCTION

The U.S. Army Ballistic Research Laboratory (BRL) recently constructed a *two-degree of freedom* mount suitable for slewing loads up to 800 pounds. This work was part of the LABCOM TEAM COOP Program involving, in addition to BRL, the Harry Diamond Laboratory and the Human Engineering Laboratory (CY525 Intelligent Ramping Stepper Motor Controller 1984). The mount was controllable in both azimuth and elevation and could be used as a weapons platform or as a surveillance and tracking platform. In order to expedite the use of the mount, a single card computer (SCC) was designed, constructed, and programmed by the author which served as a controller and communications interface for the mount. The SCC was connected by an RS232 data link to a PC class computer, termed the *host computer*, that allowed the control of the mount by a remote operator.

This report describes both the SCC, its software and, in addition, the software controlling host computer used by the operator, also written by the author. Diagrams of the SCC and pinouts of all the major integrated circuits will be included along with listings of the software in Appendices A, B, and C.

2. OVERVIEW OF THE SCC

The SCC was designed to control a mount consisting of an azimuth axis and an elevation axis. For each axis a 13-bit shaft angle encoder for determining mount angle was used along with a stepper motor, an intelligent stepper motor controller, and a corresponding stepper motor amplifier. The SCC, or simply the mount computer, required a serial data link to the host computer which input commands could be entered by the operator. The design philosophy behind the mount computer was that it would support serial communications to and from the host computer using a "high" level language to convey position and status information. The mount computer was also designed to support a "low" level language to allow communications to and from the two intelligent stepper motor controller integrated circuits (ICs) that were incorporated directly within the mount computer.

The mount computer's software was designed to take as much of the computational burden as possible off of the host computer, allowing the host computer to communicate only high level

information. Information for positioning the mount takes the form of an axis designator, an angle to be acquired, and the actual slew or "go" command. Upon receiving a command, the mount computer indicates a busy condition for the designated axis as long as slewing about that axis is taking place.

The stepper motor controllers used were CY525 Intelligent Ramping Stepper Motor Controllers (Zilog Z80-CPU Technical Manual 1976) manufactured by Cybernetic Micro Systems, Inc. (1984), San Gregorio, California. The CY525 stepper motor controller is packaged in a 40-pin dual-in-line IC. It communicates commands and data through an 8-bit-wide data bus. The CY525 generates four output phases for controlling a four-phase stepper motor and an equivalent single output phase and corresponding direction output for driving an "intelligent" stepper motor amplifier. Other outputs include a busy*/ready status, a slew* status, and a run* status. (In the text an "*" after a logic definition denotes that the logic function is active low.) Inputs include handshake logic lines for the input and output of information over the data bus such as i/o requests.

Communications to and from the CY525 can either be in the form of encoded ASCII-decimal or binary, with binary being selected for this application. The CY525 instruction set has 27 commands such as looping, branching, testing status conditions and various executables. Because of its richness of commands, the CY525 can be programmed to execute complex and lengthy programs on its own. In this application, the CY525 was not used in the "execution" mode, but was setup to execute commands one at a time as they were received. As a consequence, only of a few of its possible commands were utilized.

The principle advantage of using a stepper motor controller IC is that it allows for the automatic computation of stepping rates as a function of the number of steps within the constraints of its operational parameters. Thus, by programming the CY525 with various operating limits such as the maximum ramp rate rates for acceleration, maximum step rate, and initial ramp rate, the CY525 can compute a stepping sequence based on a "least time to turn." All that is required is to program the various limits and then to send to the CY525 the number of steps, the direction the stepper motor is to turn, and a go command, which is one of the CY525's executables.

Normally, the programming of the operational parameters of each CY525 is done each time the mount computer is reset. The mount computer then waits for the host computer, at which time it generates a low level request to the appropriate CY525. For the mount computer to control a CY525, it must first determine the initial angle of the corresponding axis. Once the initial angle is known, the mount computer converts the information about the terminal angle into a direction and number of steps. The number of steps is simply the angle for the mount to be turned through in degrees times the number of steps required by the stepper for each degree of angle. The angle to be turned through, of course, is simply the final or desired angle minus the initial angle where the initial angle, as mentioned, is determined by reading a shaft angle encoder prior to the slew operation.

In determining the initial angle, the mount controller must "read" the appropriate shaft angle encoder and convert the resulting binary information into an angle in degrees. Since the angle the mount turns through to produce one turn of the shaft angle encoder is known, each reading of a shaft angle encoder can be converted into an angle by a suitable scaling.

The mount computer requires various types of input/output (i/o) elements since it must interface to two shaft angle encoders, two CY525s, and a host computer. Communications to the host is done serially through an RS232 serial communications interface, a standard interface used frequently to connect computers with terminals. The shaft angle encoders and the CY525s are accessed through parallel interfaces since the speed for parallel i/o is far greater than for serial. Since the mount computer receives high level commands from the host, it must have a CPU (central processing unit) to convert these high level commands into low level commands suitable for the CY525s. In addition, the CPU must "read" the appropriate shaft angle encoder and compute the number of steps and the corresponding direction for each slew request. That is, the CPU must be able to communicate utilizing various i/o devices and to be able to perform all necessary operations in, essentially, real time.

The CPU must be able to "parse out" the high level commands received from the host computer and generate appropriate low level commands for the CY525s. It then must "present" these commands to the CY525s in the proper order. The CPU must also monitor the operations of the CY525s and generate any diagnostics should busy or fault conditions occur. Because of the need to store pointers, intermediate results, etc., the mount computer is required to have a

RAM (random access read/write memory) as well as an EPROM (electronically programmable read only memory) for program storage.

3. DETAILED DESCRIPTION OF THE SCC MOUNT CONTROLLER

The CPU that was selected for the mount computer was the Zilog Z80-CPU microprocessor, manufactured by the Zilog Corporation, Cupertino, California. The Z80 has, over the years, become the most widely used 8-bit microprocessor of all for process control applications. The Z80, as most if not all microprocessors, utilizes three buses: a data bus, an address bus, and a logic or control bus. The data bus defined by the Z80 is 8 bits wide. The address bus, on the other hand, is 16 bits wide and is able to support an address space spanning 65,536 bytes of memory, each byte being separately addressable. In hexadecimal notation, the address space defined by the Z80 is from 0000H to 0ffffH or from 0 to 65535 in decimal. The logic or control bus consists of a system control portion and a CPU control portion. Because the control bus is normally not compatible with the various memory or i/o elements used in the mount computer, a number of "glue" logic elements must be used to allow the CPU to utilize and control both the memory and i/o elements. In designing computers, it is essential to not only know the function of all the pins on the CPU chip but also their timing sequences as well as the timing sequences for the i/o and memory ICs. The Zilog Z80-CPU Technical Manual (1976) defines the function of each pin of the Z80 and describes its timing sequences (for example, those for memory fetches, for i/o operations, etc.).

Each i/o and memory element in the mount computer must interact in some way with the three buses. In some instances address and logic functions are combined into "chip select" and "chip enable" functions, which is done by the glue chips. All elements though, be they memory or i/o, connect to the data bus directly using tri-state logic. The data bus is a bi-directional bus in which information can be passed either to the CPU from the various support elements or to the various support elements from the CPU. As a consequence, the data bus must employ tri-state logic that assures that only one element at a time can take control of it for transferring information. The control of the data bus, in terms of who can drive it, is always maintained by the CPU. The CPU "selects" various support elements as needed and yields control for short periods of time to those elements. As a consequence, each support element must have a unique identification or memory space. If the CPU requires information from a specific memory location, only the appropriate

element must respond. As a consequence, the glue chips must decode information from the address bus and provide suitable "select" logic signals to the appropriate memory chip. The same is true for the i/o elements.

A description of the mount computer therefore must begin with a "pinout" of each IC, be it either a memory chip or an i/o chip, and a pinout of the CPU itself. Each pinout must define logic pins, data pins and, if used, memory address pins. Later in this report, descriptions of how logic level signals derived from the address bus and the control bus are combined by the glue chips to provide the chip select and chip enable functions will be given.

Once the pinouts are defined, the functions of the memory and i/o elements and their address space must be defined as well as how they are connected to the CPU, the host computer, the shaft angle encoders, the CY525s, and the glue chips. Central to the chip select and enable functions is the use of a 74154 TTL (transistor to transistor logic) compatible 4-line to 16-line decoder/demultiplexer.

The elements used include the 8251A programmable communication interface chip that supports serial communications and the 8255A programmable peripheral interface chip that supports parallel i/o. The memory elements used include the 6116 16,384-bit (2048x8) static CMOS RAM and a 2716 16,384-bit (2048x8) UV erasable CMOS EPROM. The glue chips consist of standard TTL operations, such as the "and," "or," and "not" operations. For example, the "7400" TTL chip provides four 2-input positive nand (not and) gates, etc. In addition, the mount computer utilizes a MC1488 quad line driver for driving the RS232 serial interface data link for communicating with the host computer and a MC1489 quad line receiver for converting RS232 logic level signals from the host computer to TTL level signals.

4. TECHNICAL DESCRIPTIONS AND PINOUTS OF THE CPU AND SUPPORT ELEMENTS

Because of the many and complex details of the machine cycle timing diagrams of the CPU, detailed timing charts were not included. In general, timing diagrams give the time during an instruction execution when memory or i/o elements can "read" information on the data bus or when they can assume control of the bus and place information on it for the purpose of writing. It is important to remember that timing is based on clock cycles ("T" states) within each instruction

execution cycle. Each instruction requires a specific number of T states, the first defining the actual instruction fetch. A smaller number of T states are required if the instruction to be executed is a logical operation, as opposed to an input or output operation. All glue chips as well as i/o and memory elements must operate fast enough to keep up with the system clock. Figure 1 gives the pinout and pin definitions for the Z80 CPU.

Neither the M1* pin, the CPU bus control pins, or the CPU control pins defined in Figure 1 are used by the mount computer. Of the system control pins, only the output pins MREQ*, IORQ*, RD*, and WR* are used. MREQ becomes active when the CPU is performing either an input or output operation to a memory element. IORQ, on the other hand, becomes active only if the CPU is performing an input or output operation to an i/o element. RD is active if the CPU performs any input operation, either from an i/o element or a memory element, and WR is active only if the CPU is performing an output operation.

The second pinout and pin description are for the 8255A Programmable Peripheral Interface chip and are shown in Figure 2.

The 8255A is used in the mount computer to provide parallel input/output operations. Four 8255As are used, one each for the two shaft angle encoders and one each for the two CY525s. The 8255A contains three 8-bit ports (designated as A, B, and C), making a total of 24 programmable i/o lines altogether. In addition to the 8-bit data bus that allows data communication with the CPU, the 8255A contains a CS* (chip select*) pin, an RD* (read input*) pin, a WR* (write output*) pin, and a reset pin. Any of the three i/o ports, A, B, or C, can be selected by the use of the A0 and A1 pins which are binary-input port-select pins. A0 is connected to the least significant line of the address bus and A1 next to the least (that is, standard address decoding is used by the 8255A). After a reset, the three ports are assumed to be in the read or input mode and must be programmed by the CPU before they can be used in the output mode. Many options and combinations of input and output functions for the these three ports can be programmed through software. The programming of the four 8255As will be outlined in the portion of the report describing the software.

The third pinout and pin description is for the 8251A Programmable Communication Interface and is shown in Figure 3.

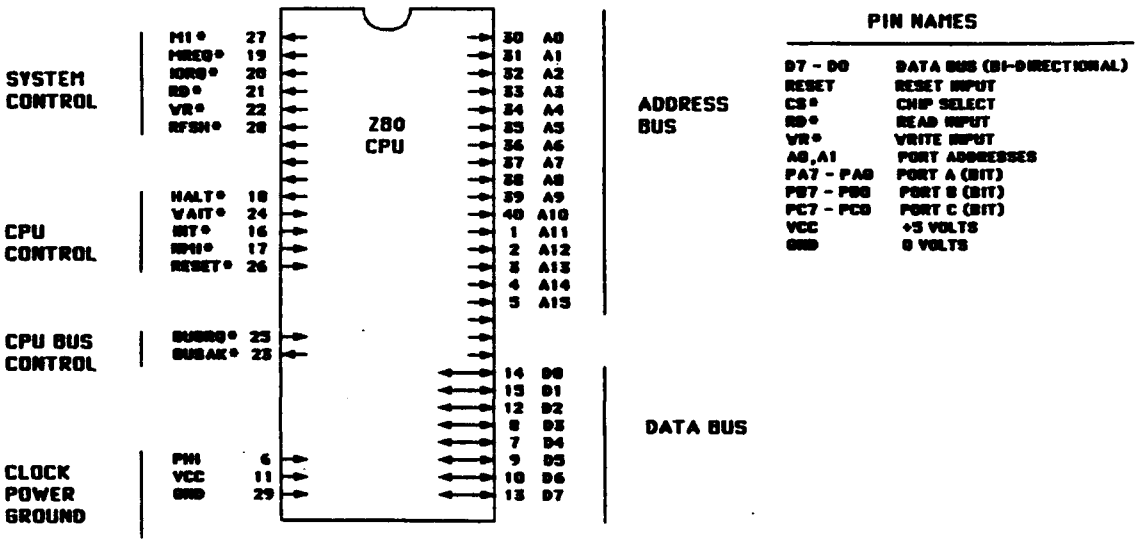

Figure 1. The Z80 CPU Pinout and Pin Description Diagram. The address bus and the data bus are shown to the right, and the control bus, composed of system control, CPU control, and the CPU bus control is shown to the left. The clock signal, applied to pin 6, is a 2.4576-MHz square wave generated by a crystal-controlled oscillator. M1*, FRSH*, HALT*, INT*, NMI*, BUSRQ* and BUSAK* are not used. RESET* is an input. Only the logic signals MREQ*, IORQ*, RD*, and WR* are used with the mount computer.

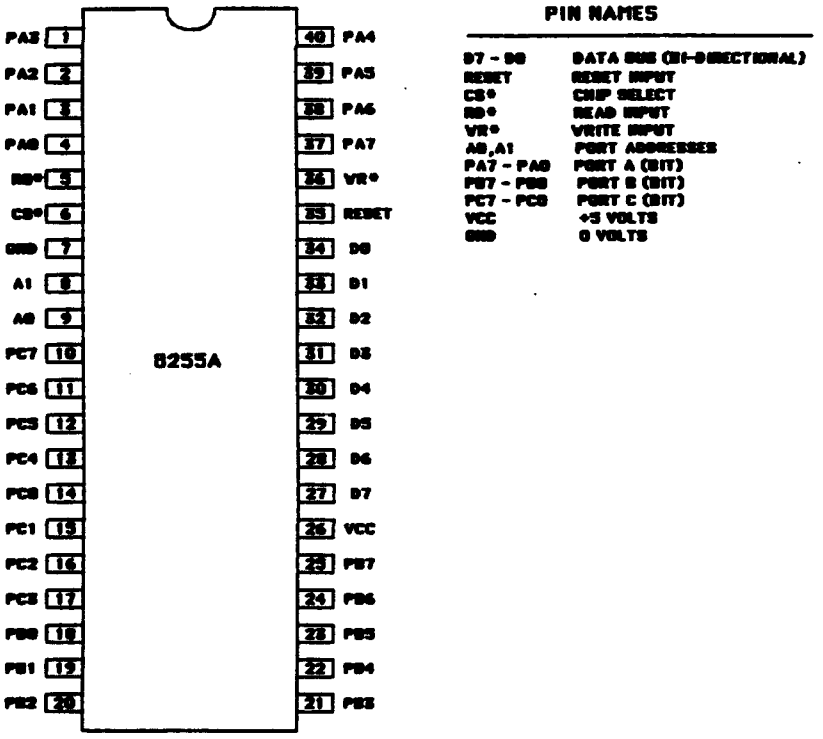

Figure 2. Pinout and Pin Description of the 8255A Programmable Peripheral Interface Integrated Circuit. The 8255A is used in the support of parallel input/output operations and contains 24 i/o lines that can be programmed in blocks as either input lines or as output lines.

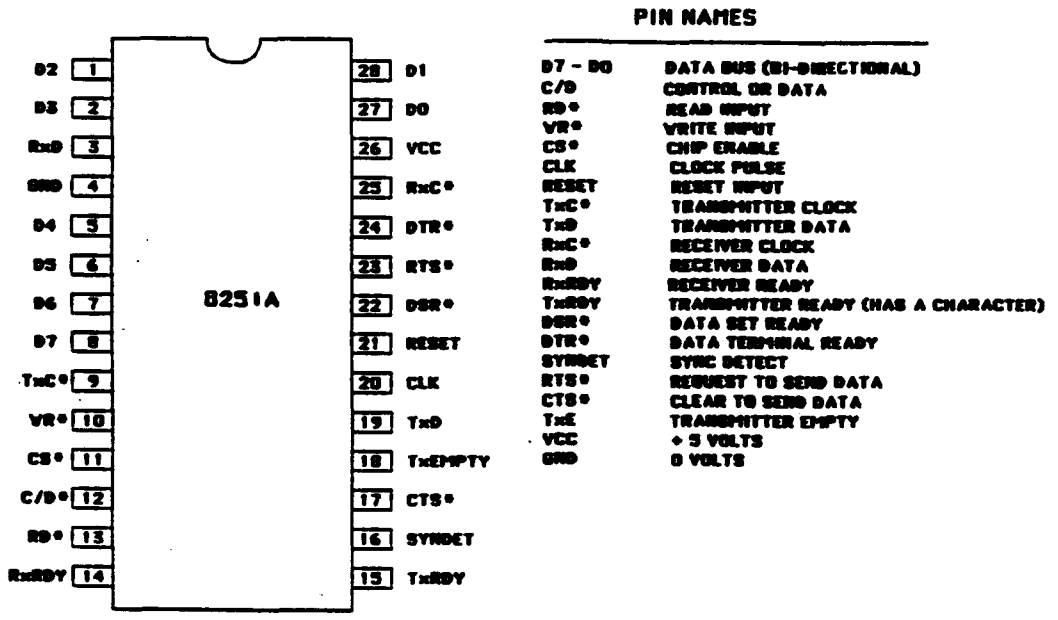

Figure 3. Pinout and Pin Description of the 8251A Programmable Communication Interface Chip. The 8251A supports the serial communications to the host computer. Baud rates for both transmitting and receiving are determined through software and an external clock, derived from the system clock by dividing by powers of two.

The 8251A programmable communication interface is used to generate TTL level signals for transmitting serial information between the mount controller and the host computer. Conversion of the TTL level output signal to an RS232 level is performed by one section of an MC1488 quad line driver. One section of an MC1489 is used to perform the reverse process, of converting from the RS232 logic level to the TTL level. The 8251A can provide both synchronous and asynchronous operation. In the asynchronous mode of operation, as it is used in the mount computer, it supports a format consisting of a start bit, seven data bits of information, and two stop bits at 19.2 kHz baud. The 8251A is termed a two-port device in that two port addresses are required for its operation. The first port is used for programming and for checking operating status conditions. The second port is used to input characters received by the unit or to output characters that are to be transmitted. The 8251A has a CS* (chip select*) pin, an RD* (read data command*) pin, a WR* (write data or control command*) pin, a reset pin, a clock input pin, as well as several pins denoting the status of the various communications. Serial output and input are at the TTL level, and each is converted to RS232 logic levels as needed by the MC1488 and

MC1489. The circuitry and the glue chips required to utilize the 8251A will be described later in this report.

The final two major elements of the mount computer are the 6116 RAM and the 2716 EPROM. Both of these memories have the same pinout and pin definitions. Because these are memory elements, they both require address information from the address bus. They utilize only enough address information, though, to span the address space they need. In addition, they both utilize chip enable and output enable pins. Of course, they both utilize the data bus, as does the i/o. A pinout and pin definitions are given in Figure 4.

Both the 6116 RAM and the 2716 EPROM utilize 11 address lines (out of the 16 supported by the Z80). In addition, each chip utilizes a chip enable* pin and an output enable* pin. Both memory elements utilize a single +5-V power source. Programming the 2716 EPROM is done separately from the mount computer by a special EPROM programming device from data files generated by an assembler. A listing of the assembly language program will be included in the Appendix B of this report.

The next integrated circuit to be discussed is the CY525 stepper motor controller. Though the CY525 contains a data bus and various control bus pins, they are not connected to the data control bus of the mount controller. Instead they are connected to eight i/o pins of an 8255A which serves an interface function. Because the CY525 is embedded, many of its control and status pins are not used, such as outstrobe*, clock/15, dowhile, etc., while others, such as the ascii/binary* pin, are hard-wired either high or low. By suitably connecting the CY525, it could in fact be made to work from a keyboard, though that is not done here. The principle reason for using the CY525 in the mount computer is because of its ability to perform ramp computations in real time. A more detailed description of the CY525 stepper motor controller chip can be found in a publication from Cybernetic Micro Systems, Inc., its manufacturer (Ref. 2). The pinout and pin definitions of the CY525 are given in Figure 5.

5. DETAILED DESCRIPTION OF THE MOUNT CONTROLLER

This portion of the report will concentrate primarily on the control portion of the mount computer. As indicated, the data bus runs from element to element to element, linking all the

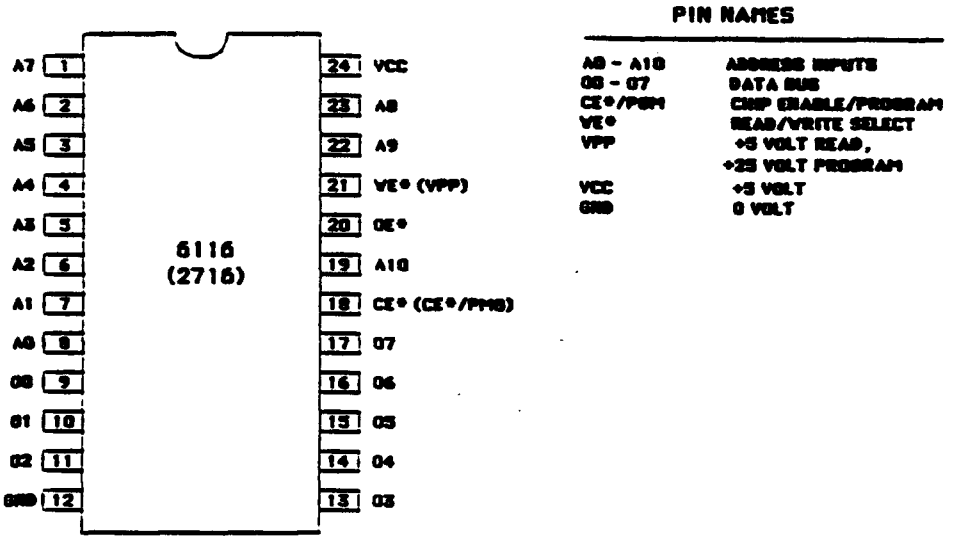

Figure 4. Pinout and Pin Definitions of the 6116 16,384-bit (2048x8) Static CMOS Random Access Memory and 2716 Electrically Programmable Read Only Memory. The pinout for both elements are identical as far as the mount computer is concerned. Elements in parentheses apply to the 2716 EPROM.

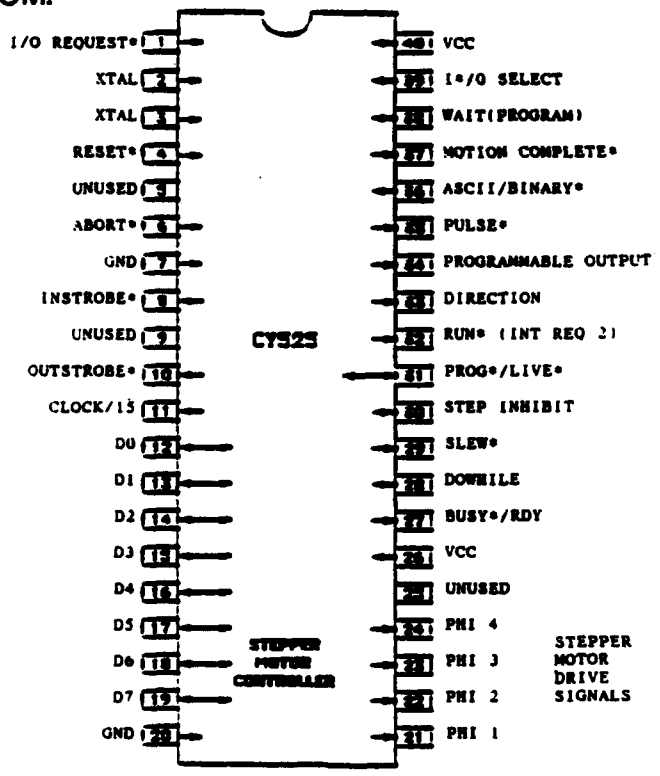

Figure 5. Pinout and Pin Definitions of the CY525 Intelligent Ramping Stepper Motor Controller. An 8.0-MHz crystal is connected between pins 2 and 3. Output from the CY525 for driving a stepper motor was from pulse* (pin 35) and direction (pin 33).

elements that utilize the data bus, such as the CPU, i/o, and the memory. To a lesser extent, the same is true for the address bus. For the mount computer though, only 11 of the 16 address bus lines are used. The control bus portion, on the other hand, requires a number of glue logic chips to generate the proper selects and enables.

The clock circuit used in the mount controller is constructed using two TTL sections from a 7404-hex invertor chip, two 2.2-k resistors, a 0.001-μf capacitor, and a 4.9165-MHz crystal. Output from the clock is fed into a binary counter composed of one section of a 7493 4-bit binary counter. The counter serves to "clean up" the clock signal and to provide a 50% duty cycle. It does this by changing its output only on positive transitions of the clock. Figure 6 shows how the clock circuit is wired.

With regards to device or element selection, four control output pins on the Z80 are utilized, as mentioned previously. There is a pin to indicate that the Z80 is performing an i/o function, termed the IORQ* pin; one used when performing a memory operation, the MEMR* pin; one used when an input function is to be performed, be it either i/o or memory, the RD* pin; and one used when an output function is performed, the WR* pin. The glue logic takes these control signals, along with portions of the address bus, and generates the appropriate select and enable pulses for all the i/o and memory. Figure 7 defines 16 additional TTL logic level lines, S0* through S15* (of which only 5 are used), which are device select lines derived from IORQ* and the 4 address lines (A2 through A5) by the 74154 4-line input 16-line output demultiplexier.

All the logic shown in Figure 7 is active low, and, as a consequence, all the necessary chip select pulses for i/o operations can be obtained directly from the 74154. Since the 74154 utilizes as inputs address lines A2–A5, each i/o element is assigned four address spaces. An i/o element can only be selected if the IORQ* pin of the Z80 is low, indicating that an i/o operation is taking place. Specific addresses within the assigned block of four is specified by A0 and A1 while the i/o function is specified by either the RD* pin of the Z80 or the WR* pin.

Unlike the i/o elements, the memory elements utilize the address bus directly, being connected to A0 through A10. The next higher address line, corresponding to A11, is therefore used as a chip select line. By feeding A11 directly to the CE* pin of the 2716 EPROM, an address space of 00000h to 07ffh can be assigned, since A11 must be low for the 2716 to be

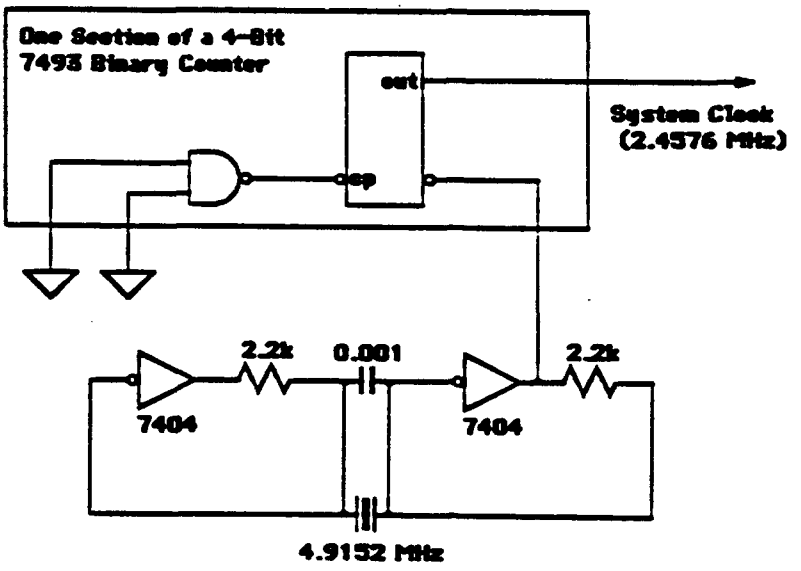

Figure 6. Clock Circuit for the Mount Computer. A standard oscillator circuit utilizing two sections of a hex invertor is used to produce a "quasi" TTL clock output at 4.9152 MHz. One section of a 7493 4-bit binary computer divides the rate by two and gives a true TTL output with a 50% duty cycle.

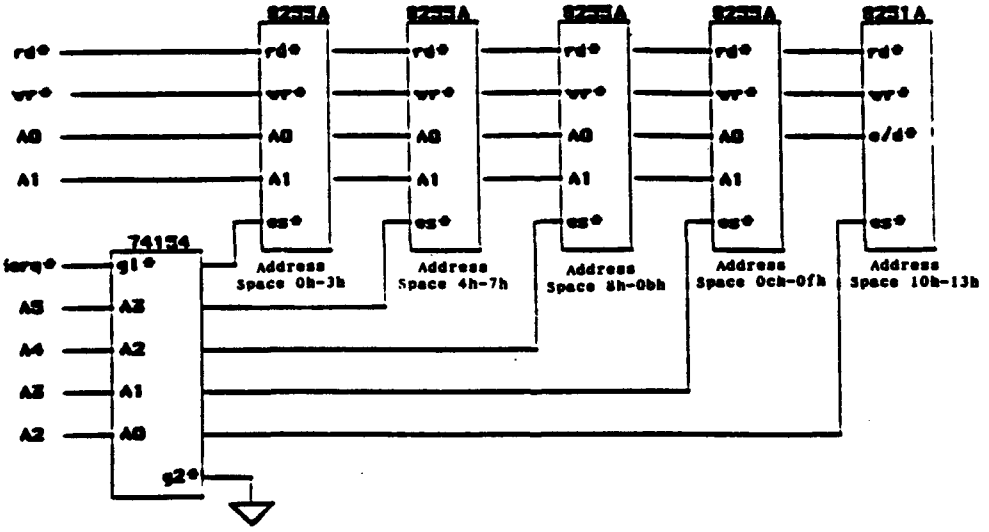

Figure 7. Schematic Diagram Showing the Glue Logic Chip for Generating Chip Select for I/O Operations. A 74154 4-line to 16-line decoder/demultiplexer is utilized. Each i/o element is assigned four address spaces. The first (left most) 8255A services the azimuth shaft angle encoder, the second the elevation shaft encoder, the third the azimuth CY525 controller and the fourth the elevation CY525 controller. The 8251A services the serial interface connections that link the computer to the host.

enabled. On the other hand, by feeding A11* to the CS* pin of the 6116 RAM, an address space of 0800h to 0ffffh can be assigned, since A11 must be high for the 6116 to be selected. The select circuitry of the memory ICs is shown in Figure 8.

Selecting and enabling the 2716 EPROM is accomplished by wiring the RD* line ("or"ed with the MREQ* line from the Z80) to the OE* (output enable*) pin of the 2716 and the A11 (address bus, bit 11) line of the Z80 to the CE* (chip enable) of the 2716. Selecting and enabling the 6116 RAM is accomplished by wiring the MREQ* of the Z80 to the OE* of the 6116, the WR* of the Z80 to the WE* of the 6116, and A11* of the Z80 to the CS* (chip select) pin. A11*, for example, is obtained by negating A11 by the use of a section of a hex invertor.

The memory space of the 2716 EPROM is established by requiring A11 to be at logic level 0. Any bit pattern in the address space of xxxx,0000 0000,0000B (16-bit binary) to xxxx,0111 1111,1111B will select the 2716 EPROM. (Here the "x" denotes a "don't care" bit.) The memory space of the 6116 RAM, because of the inversion of A11, is from xxxx,1000 0000,0000B to xxxx,1111 1111,1111B. For programming, it is therefore assumed that the address space of the 2716 is from 0000H to 07ffH and that of the 6116 from 0800H to 0ffffH.

It is essential that the 2716 EPROM be addressed starting at 0000H because after a reset the program counter of the Z80 is set to that address. That is, the first instruction to be executed by the Z80 after a reset operation is at address 0000h.

The interface or electrical connections between an 8255A programmable peripheral interface i/o element and a CY525 stepper motor controller integrated circuit is somewhat different than, for example, a memory element and the Z80 since the CY525 is considered a peripheral and not located directly on the data bus. Because of this, the CY525 requires both a hardware interface as well as a software interface. The hardware interface is shown in Figure 9.

In Figure 9, PC4 serves as a strobe input*, latching eight data bits from the CY525's parallel data bus into the 8255A's port A. A "ready" condition is set through PB5, indicating that valid data is on the parallel data bus to be input by the 8255A. Output of 8-parallel data bits to the CY525 is accomplished by placing the data on port A and strobing PC0 (connected to the i/o request* pin) low while PC1 (i*/o request pin) is low. The motion complete* pin is read to determine when a slew command sent to the CY525 is completed.

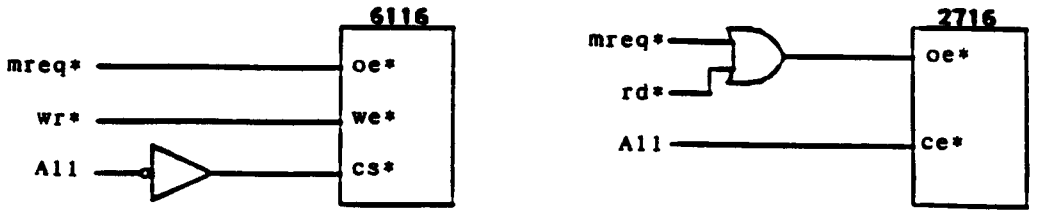

Figure 8. Memory Enable Circuitry of the Mount Computer. The two input "or" section prevents inadvertent writes to the 2716 EPROM.

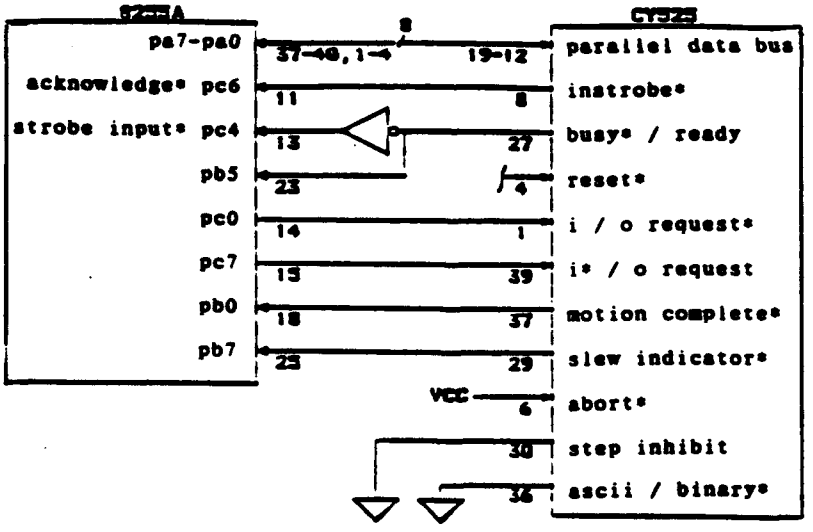

Figure 9. Interface between an 8255A Programmable Peripheral Interface Input/Output Element and a CY525 Stepper Motor Controller. The 8255A is used in operating mode "two" in which port A (pins PA7–PA0) is bi-directional and where PC0 and PC1 are set and reset under software control through the "single bit set/reset" feature of the 8255A. Port addresses of 08h–0bh are used for the CY525 that controls the azimuth stepper and 0ch–0fh for the CY525 that controls the elevation stepper.

The input of information from a shaft angle encoder is far simpler than for the case of a CY525. For the input of information from a shaft angle encoder, all that is required is the "hold" pin of the shaft angle encoder to be held low for 2 μs. Information is "frozen" and placed on the 13 output pins of the shaft angle encoder which can be input by an 8255A using 13 parallel input lines. Since more than eight data bits are input, two input ports are required. Figure 10 defines the interface between a 13-bit shaft angle encoder and a 8255A parallel input interface element.

The pin definition and interface connections of the shaft angle encoder used in the support of elevation measurements is shown in Figure 11.

6. DESCRIPTION OF THE SOFTWARE USED IN THE SUPPORT OF THE MOUNT COMPUTER.

The software for the mount computer is written in assembly language using Zilog mnemonics. In the program, symbols are used which are composed of one or more alphanumeric characters and the underscore. Among other things, symbols are used to denote variables. When parentheses are used around symbols defined as variables, they denote that the address associated with the symbol is to be inferred. Symbols followed by a colon denote an address label. Characters following a semicolon denote comments.

In order to make the logic behind the assembly language more intelligible, a "pseudo" program has been written using a cross between the English language and Pascal. As such, it is not only suggestive of the actual assembly language program, but it can be understood far more easily. Regrettably, in order to understand the programming of the i/o elements, one must refer to component data catalogues that list the specific elements. The assembly language program, though, should serve as a example for specific cases.

The variables in the assembly language listing as well as the pseudo listing include channel(8), n_o(8), n_p(8), g_o(8), g_p(8), n_lo_o(8), n_lo_p(8), sae0_o(16), sae_p(16), sae1_o(16), sae1_p(16), cc(16), and others, as listed. Subscripts after a variable denote an array. The variables contain either numerical values or flags. Control of the program is maintained by the use of flags that are tested, set, and reset as needed.

Itek uS/23K Shaft Angle Encoder			8255A Input/Output Interface Element		
Function	Pin Number		Function	Pin Number	Port Address
VCC	1				
Bit 0 (LSB)	2	→	PB0	21	1
1	3	→	PB1	22	1
2	4	→	PB2	23	1
3	5	→	PB3	24	1
4	6	→	PB4	25	1
5	7	→	PC0	14	2
6	8	→	PC1	15	2
7	9	→	PC2	16	2
8	10	→	PC3	17	2
9	11	→	PC4	13	2
10	12	→	PC5	12	2
11	13	→	PC6	11	2
12	14	→	PC7	10	2
GND	15				
Hold	16	←	PA0	4	4

Figure 10. Pin Definition and Interface Connections of the 13-bit Shaft Angle Encoder in Support of the Azimuth Measurements. "Data freezing" occurs 2 μs after the "hold" pin of the shaft angle encoder is brought low. After reading or inputting the data, the hold pin is returned high by PA0.

Itek uS/23K Shaft Angle Encoder			8255A Input/Output Interface Element		
Function	Pin Number		Function	Pin Number	Port Address
VCC	1				
Bit 0 (LSB)	2	→	PB0	21	5
1	3	→	PB1	22	5
2	4	→	PB2	23	5
3	5	→	PB3	24	5
4	6	→	PB4	25	5
5	7	→	PC0	14	6
6	8	→	PC1	15	6
7	9	→	PC2	16	6
8	10	→	PC3	17	6
9	11	→	PC4	13	6
10	12	→	PC5	12	6
11	13	→	PC6	11	6
12	14	→	PC7	10	6
GND	15				
Hold	16	←	PA1	3	4

Figure 11. Pin Definition and Interconnection of the 13-bit Shaft Angle Encoder Used in Support of Elevation Measurements.

Though the program is set up to handle only two channels, in theory 256 channels could be handled by simply cloning sections of the program for each additional channel. A typical slewing command issued by the host would take the form of a channel specifier, an "N" command denoting that the following two characters are to be interpreted as data bytes, and a "G" command for initiating mount motion of the specified axis to the prescribed angle. A single command is the "ping" command. When the host sends a ping command, the mount computer immediately responds with an "ack." This lets the host know that the mount computer is "alive." There is also a "g_status" command used by the host for determining which, if any, of the mount's axes are busy. The use of the g_status command precludes the sending of additional commands to the mount computer while it is still busy.

Because of the rather simple command structure, the host is not able to change the ramping parameters of the CY525s. All ramping information is programmed into the mount computer's EPROM and is output to the CY525s only on power up or after a hardware reset. The programming of all the i/o elements also occurs at the very same time, even before the programming of the CY525s, since the CY525 programming information must come through an 8255A. Baud rates associated with the 8251A likewise cannot be changed but are also fixed in the EPROM.

Upon reset, various "boolean" variables are set to false to indicate a no operation condition. As commands are received from the host, the mount computer updates these variables to true as needed. Entry into various parts of the program is controlled by these variables used as flags. After a pending task is accomplished, the corresponding flag is set to false, denoting completion of the operation. Another variable, "channel," is used as a pointer to the currently assigned channel, be it the azimuth channel or the elevation channel. All information from the host is directed to the active channel, of which there can be only one. The program must distinguish between data and such things as command and channel specifiers. This is accomplished by establishing a protocol in which all input is taken as commands until the "N" command is sent. In this case, after the receipt of the N command, the next two bytes are assumed to be an angle specified, even though these data bytes may be valid commands or channel specifiers.

The "G" or "go" command will only be accepted after an N command and two data bytes, at which time the selected axis of the mount will begin to slew. During the slew command, the

active channel is "unavailable" until the corresponding CY525 signals that motion is complete. Commands sent to an active channel will be ignored, but any nonactive channel can be programmed. Thus, it is possible to have multiple channels operating at the same time.

Appendix A contains a listing of the pseudo control program whose basic purpose is to merely indicate program flow and to aid the reader in understanding the more detailed assembly language listing. Both the pseudo program and the assembly listing use the same statement labels. These programs consist of three sections—an initialization and program control section, a section devoted to the "o" or elevation channel, and a section devoted to the "p" or azimuth channel. Here "o" and "p" are used to denote channels as opposed to the use of the words "elevation" and "azimuth." The symbol "reg_c" denotes the "c" register of the Z80 CPU. As can be seen in the assembly language listing in Appendix B, the h and l registers of the Z80 are used quite extensively as they can be combined into a single 16-bit register in support of 16-bit arithmetic.

The angle "space" in both azimuth and elevation is spanned by a twos compliment 16-bit representation. As a consequence, care has to be taken in computing angular differences to prevent overflow, since all possible bit patterns may be used to represent angles. Because gear reduction units are used to turn the shaft angle encoders, their output must be scaled. In addition, the terminal angles must be scaled in terms of the number of steps required by the stepper motors. Again, care must be taken to assume that the precision limitations of the CPU (that is, representing angles in a 16-bit format) are allowed for. It may also be noted that the mount computer does not use interrupts, since all activity occurs on time scales commensurate with machine or human time scales as measured in seconds.

Upon receipt of a channel specified and a "N" command, the appropriate n_x and n_lo_x variables are set true. Here "x" denotes either an "o" or a "p." For the sake of discussion, assume that the "o" channel has been selected. Upon receipt of the next character from the host, a data byte denoting the least significant portion of the angle, the program is directed to line l14 where n_lo_o is tested to see if it is true, which it is in the case of the first data byte. The program then places the data from the host into n_v_lo_o and immediately sets n_lo_o false. After the next character is received, the program is again directed to line l14, but this time it is redirected to l17 because of n_lo_o being false. The new data is now placed into n_v_hi_o and

n_o is made false. At this point, the appropriate shaft angle encoder is input, number of steps and direction computed, and the CY525 programmed. All that is now needed for the CY525 to generate stepper pulses is the receipt of a "go" command from the host.

Upon receipt of the next character from the host, the program falls through to line l9 where a test is made to see if the character is a "G," denoting "go." If it is, then a "go" command is transmitted to the CY525 stepper motor controller, and a "G" character is sent to the host, indicating the stepper response. At this point, the status of the CY525 is continually tested following line l1_o. Status information is made available to the host at line l1_ae. New input in the form of an "N" command could be sent to the mount computer at this time and would be accepted since g_o is merely a toggle. However, the operation of the software in the host precludes this option, waiting for the stepper to complete its operation before additional commands are transmitted. Thus, the host is assured that all commands will be executed and none lost. Because the transmission time for the transmission of instructions is short compared with a typical slew time interval, the CY525 could essentially be kept busy all the time if needed.

7. DESCRIPTION OF THE SOFTWARE IN SUPPORT OF THE OPERATION OF THE HOST COMPUTER.

As mentioned previously, the host computer is equivalent to a PC clone and is in the form of a Z80-based Digilog microcomputer. One reason for selecting the Digilog was that several were on hand and that assembly language "hooks" were available for supporting input and output operations that could be invoked from Pascal programs. Assembly language segments can be placed in a Pascal program by the use of the Pascal "inline" statement. By the use of the Pascal "var" that allows variable declaration in procedures and functions, program variables can be incorporated into the assembly language modules.

A listing of the Pascal program utilized in the host computer is given in Appendix C. One of the objectives of the program is to perform diagnostics on the remote computer whenever possible. After a command is sent to the remote computer, the host times the response from it. If the response time exceeds a preset limit, then a diagnostic message is displayed to the operator. In some instances, helpful suggestions are included in these messages.

After a reset, a "ping" message is sent to the mount computer. If a proper response is not received within the preset time limit, the program displays a diagnostic message. The program also prevents efforts on the part of the operator to send mount messages. Only when a proper response from a ping is received will the operator be queued for input. After each character of a command is sent, the computer waits for an "ack," signifying that the remote has accepted the character and is ready for another. After the issuance of the go command (the "G" character), the host periodically checks the status of the remote to determine when the corresponding slewing command sent to a CY525 is completed.

By the use of such checks, the proper operation of the remote computer is more nearly assured. Synchronization is also maintained between the two computers during the transmission of an instruction.

8. SUMMARY

This report describes both the hardware and the software of a two-axis mount controller computer and the associated drive software of a host computer. Included were descriptions of the various computer elements used, such as the CPU, i/o, and memory elements. A simple hardware design was selected based on the ubiquitous Z80 CPU manufactured by the Zilog Corporation. The program for the mount computer was stored in an EPROM that was programmed from assembly language using a Zenith Data Systems computer and an EPROM programmer board. Software for the host computer, used as a control terminal for the mount, was written in Pascal and stored on a floppy disk. During execution, the control program in the host resided in its random access memory, being read in from the floppy disk drive.

It is hoped that the documentation included in this report will enable others to replicate the mount computer, including all the software, for similar applications. It is also hoped that this report will give the reader a deeper insight into the use and operation of microcomputers in general.

The hardware and software shown were was the simplest and quickest to develop, allowing use of the mount while a more complex controller was being developed. Because of this

simplicity, bugs in the software and in the mount computer itself could easily be fixed, allowing developmental efforts to focus on the mount itself.

9. CONCLUSION

The mount controller computer, its software, host software, and serial interface link between the two computers have worked extremely well. No evidence of a failure during operation was ever traced to a software fault. Occasionally, a hardware failure of either the remote mount computer or host would cause a synchronization problem where the two programs would loose track with each other. Normally, resetting both systems was sufficient to cure any such problems.

The hardware and the software both served as bread boards for system development with both being responsive to developmental needs. The success of the system provided guidance for the installation of a more complex system based on state-of-the-art process control computers costing perhaps in excess of 100 times as much.

INTENTIONALLY LEFT BLANK.

10. REFERENCES

CY525 Intelligent Ramping Stepper Motor Controller. San Gregorio, CA: Cybernetic Micro Systems, Inc., 1984.

Zilog Z80-CPU Technical Manual. Cupertino, CA: Zilog, Inc., 1976.

INTENTIONALLY LEFT BLANK.

APPENDIX A:

LISTING OF THE CONTROL PROGRAM USING A "DESCRIPTIVE" COMPUTER LANGUAGE.

Note: The Purpose of This Listing Is to Provide Only an Indication of the Program Flow and Logic.

INTENTIONALLY LEFT BLANK.

```
start: program the 8251A serial i/o element;
s1:    initialize the two 8255As that interface to CY525s;
s2:    initialize the two 8255As that interface to SAEs;
s3:    program the i/o request* and i*/o request pins the CY525s;
s5:    program the interrogate pins on the SAEs;
s6:    program the stack pointer to the top of RAM;
s7:    program the first_rate, rate, slope and divisor functions;
s8:    exercise the programmable output pins;

s10:   set channel = "channelo";
       set destination = "dest_o;
s11:   set n_o="false";
       set n_p="false";
       set g_o="false";
       set g_p="false";

l1_a:  if there is no character from host then goto l1_aa
       else input character and place it in character_from_host;

l1_b:  if channel== "channelp" and if n_p =="true" then goto l1_aa;
       if channel== "channelo" and if n_o =="true" then goto l1_aa;
       if character_from_host =="ping" then
         begin
           send "ack" to host;
           goto l1_a;
         end;

l1_ac: if character_from_host =="channelp" then
         begin
           set channel= "channelp";
           send "ack" to host;
           goto l1_a;
         end;

       if character_from_host =="channelo" then
         begin
           set channel = "channelo";
           send "ack" to host;
           goto l1_a;
         end;

l1_ae: if character_from_host == "g_status" then
         begin
           if g_o == "true" then set reg_c = "1";
           if g_p == "true" then set reg_c = "2";
           send reg_c to host;
           goto l1_a;
         end;

l1_aa: if destination == "dest_o" then goto l1_o;
       if destination == "dest_p" then goto l1_p;

l1_o:  set destination = "dest_p";     {Servicing the "o" channel.}
       if g_o == "true" and if (bit0, port b1) == "0" then set
 g_o="false";
```

```
15:     if there is no character from the host then goto l1_a;
        if channel <> "channelo" then goto l1_b;

16:     if n_o == "true" then goto 14;

17:     if character_from_host == "N" then
          begin
             set n_o = "true";
             set n_lo_o = "true";
             send "N" to host;
             goto l1_a;
           end;

19:     if character_from_host <> "G" then goto l1_a;

112:    send "G0" to the "elevation" CY525;
        set g_o = "true";
        send "G" to host;
        goto l1_a;

114:    if n_lo_o =="true" then

          begin
            set n_v_lo_o = character_from_host;
            set n_lo_o = "false";
            send "ack" to host;
            goto l1_a;
          end;

117:    set n_o = "false";
        set n_v_hi_o = character_from_host;

118:    read "elevation" shaft angle encoder into sae0_o (13 bits);
        set sae0_o = 1101*sae0_o/1024;
        set sae1_o (low byte) = n_v_lo_o;
        set sae1_o (high byte)= n_v_hi_o;

        set cc = abs(sae1_o - sae0_o);
        if sae1_o >= sae0_o then set reg_c="+" else set reg_c:="-";
        send reg_c to the "elevation" CY525;
        set cc = cc*12160;
        send "N2" to the "elevation" CY525;
        send cc (low byte) to the "elevation" CY525;
        send cc (high byte) to the "elevation" CY525;
        send cc (high byte) to host;
        goto l1_a;



l1_p:   set destination = "dest_o";   {Servicing the "p" channel.}
        if g_p == "true" and if (bit 0 of port b3)=="0" then
        set g_p  = "false";

15_p:   if there is no character from host then goto l1_a;
        if channel <> "channelp" then goto l1_b;
```

```
16_p:  if n_p == "true" then goto 114_p;

17_p:  if (character from host) == "N" then
         begin
           set n_p = "true";
           set n_lo_p = "true";
           send "N" to host;
           goto 11_a;
         end;

19_p:  if character_from_host <> "G" then goto 11_a;

112_p: send "G0" to the "azimuth" CY525;
       set g_p = "true";
       send "G" to host;
       goto 11_a;

114_p: if n_lo_p == "true" then
         begin
          set n_v_lo_p = character_from_host;
          set n_lo_p = "false";
          send "ack" to host;
          goto 11_a;
         end;

117_p: set n_p = "false";
       set n_v_hi_p = character_from_host;

118_p: read "azimuth" shaft angle encoder into sae0_p (13 bits);
       set sae1_p (low byte) = n_v_lo_p;
       set sae1_p (high byte)= n_v_hi_p;
       set cc = abs(sae1_p = sae0_p);
       if sae1_p>= saeo_p then set reg_c= "+" else set reg_c= "-";
       send reg_c to the "azimuth" CY525;
       set cc = cc*61;
       send "N2" to the "azimuth" CY525;
       send cc (lo byte) to the "azimuth" CY525;
       send cc (high byte) to the "azimuth" CY525;
       send cc (high byte) to host;
       goto 11_a;

end.
```

INTENTIONALLY LEFT BLANK.

APPENDIX B:

ASSEMBLY LANGUAGE LISTING OF THE SOFTWARE FOR THE MOUNT COMPUTER.

Note: After Compilation, the Resulting Machine Level Code Is Programmed Into the 2716 EPROM.

INTENTIONALLY LEFT BLANK.

```
; Software to drive the BRL turret mount in azimith and elevation. ;
 org h'0800
;
cc:     dw                  ; "o" denotes azimuth.
        dw
cc00:   dw                  ; 'p' denotes elevation.
        dw
cc01:   dw
        dw
cc02:   dw
        dw
cc04:   dw
        dw
cc08:   dw
        dw
cc16:   dw
        dw
cc32:   dw
        dw
cc64:   dw
        dw
cc128:  dw
        dw
cc256:  dw
        dw
cc512:  dw
        dw
sae_o:      dw
sae_p:      dw
sae0_o:     dw
sae0_p:     dw
sae1_o:     dw
sae1_p:     dw
n_o:        dw
n_p:        dw
g_o:        db
g_p:        db
chr:        db
n_lo_o:     db
n_lo_p:     db
n_v_lo_o:   db
n_v_lo_p:   db
n_v_hi_o:   db
n_v_hi_p:   db
channel:    db  ;Controlled by host, specifies channel,"o" or "p".
chr_ready:  db      ; Boolean.
destination: db     ; Controlled locally.  Toggled locally.
;
; initialize the 8251A serial interface, V.
;
 org h'0000
;
    ld a,b'0 J01110  ;1 stop bit, no parity, 8 data bits, 16x baud
rate factor.
    out (portb5),a   ; write to the 8251A command port, mode byte.
    ld a,b'00110111  ; command instruction.
    out (portb5),a  ;write to the 8251A command port, command byte. ;
; initialize 8255A-I and 8255A-III, the two CY525s interface chips.
```

```
    ld a,b'11000010  ; Control word.
    out (portd1),a   ; port a is bi-directional, port b input, port c
output.
    out (portd3),a
;
; initialize 8255A-II and 8255A-IV, the shaft angle encoder interface
chips.
    ld a,b'10011011  ; Control word.
    out (portd2),a   ; ports b and c are input, port a is input.
    ld a,b'10001011
    out (portd4),a   ; ports b and c are input, port a is output.
;
    ld a,noioreq  ; Set (I/O REQUEST)*, bit 1 of port c.
    out (portd1),a
    out (portd3),a
;
    ld a,ioselin     ; Set I* /O REQUEST line low for CY525 "in"
    out (portd1),a
    out (portd3),a
;
    ld a,b'00000000  ; bit 0 is the sae2 "interrogate."
    out (porta4),a   ; bit 1 is the sae4 "interrogate."
;
;initialize and set parameters on CY525.
;
    ld sp,h'1000     ; set stack pointer.
;
    ld de,st_in_o ; point to string to be transmitted to the CY525.
    call  s_pr_1_o     ; do it.  String defines initial CY525
conditions.
    ld de,st_in_p ; point to string to be transmitted to the CY525.
    call  s_pr_3_p
;
    ld de,st_on  ;Exercise programmable output pin (34) on CY525_o.
    call  s_pr_1_o
    ld de,st_off
    call  s_pr_1_o
    ld de,st_on
    call  s_pr_1_o
    ld de,st_off
    call  s_pr_1_o
    ld de,st_on
    call  s_pr_1_o
;
    ld de,st_on  ;Exercise programmable output pin (34) on CY525_p.
    call  s_pr_3_p
    ld de,st_off
    call  s_pr_3_p
    ld de,st_on
    call  s_pr_3_p
    ld de,st_off
    call  s_pr_3_p
    ld de,st_on
    call  s_pr_3_p
;
    ld a,channelo
    ld (channel),a
    ld a,dest_o
```

```
    ld (destination),a
;
    ld a,false
    ld (n_o),a
    ld (n_p),a
    ld (g_o),a
    ld (g_p),a

    ld a,b'00000010
    out (porta4),a     ; Not freeze on elevation channel.
;
; &&&&&&&&&&&&&&&&&   end initialization      &&&&&&&&&&&&&&&&&

;
; @@@@@@@@@@@@@@@@   start master loop       @@@@@@@@@@@@@@@@


ll_a:
        call test_s_in      ; Returns chr and chr_ready:boolean.
        ld (chr_ready),a
        cp false
        jp z,ll_aa
;
ll_b:
        ld a,(channel)      ; Determine if "o" or "p" to receive char.
        cp channelp
        jp z,ll_a_p
;
ll_a_o:
        ld a,(n_o)          ; Char. from HOST meant for "o"
        cp true             ; If n_o true then char. is data.
        jp z,ll_aa
        jp ll_ab

ll_a_p:
        ld a,(n_p)          ; Char. from HOST meant for "p"
        cp true             ; If n_p true then char. is data.
        jp z,ll_aa

ll_ab:
        ld a,(chr)
        cp ping           ; Respond to "ping"
        jp nz,ll_ac
        ld c,ack
        call host
        jp ll_a


ll_ac:
        ld a,(chr)          ; Get character from HOST.  Is the
        cp channelo         ; data either "channelo" or "channelp"
  jp z, ch
        cp channelp
        jp nz, ll_ae

ch:
        ld (channel),a      ; Update "channel"
        ld c,ack
```

```
        call host
        jp l1_a

l1_ae:
        ld a,(chr)
        cp g_status
        jp nz,l1_aa
;
        ld a,(g_o)      ;Get motion complete status of channel 'o'.
        ld c,b'00110000     ; '0'.
        cp false
        jp z, l1_ad_o
          ld a,c
          or b'00000001       ; Set LSB for channel 'o'.
          ld c,a              ; Store status in reg. c.

l1_ad_o:
        ld a,(g_p)      ;Get motion complete status of channel 'p'.
        cp false
        jp z,l1_ad_p
          ld a,c
          or b'00000010
          ld c,a

l1_ad_p:
        call test_s_out
        cp false
        jp z,l1_ad_p
          ld a,c
          out (porta5),a
         jp l1_a

l1_aa:
        ld a,(destination)   ; Alternate "o" and "p" loops.
        cp dest_o
        jp  z,l1_o
        jp l1_p

; &&&&&&&&&&&&&&&&&&   Start of "o" loop.     &&&&&&&&&&&&&&&&&&&
 l1_o:
    ld a,dest_p
    ld (destination),a ; "destination" points to alternating loops.
    ld a,(g_o)           ; Test if "go" active.
    cp false
    jp z,l5
l2:             ; "go" active here.  Test for "motion complete."
    in a, (portb1)
; Note: "motion_complete" active high.
   and b'00000001
   jp nz,l5

l4:                   ; Motion is now complete.  Turn off "go."
         ld a,false       ; g := false.
         ld (g_c),a

l5:
     ld a,(chr_ready)      ; Is there data from host?
     cp false              ; If not abort effort and start over.
```

```
    jp z,l1_a
;
     ld a,(channel)
     cp channelo
     jp nz,l1_b ;Check other channels if character not needed here.
16:
     ld a,(n_o)
     cp true
     jp z,l14
17:
     ld a,(chr)
     cp h'4e          ; chr = 'N' ?
     jp nz,19
18:                   ; Last character received was 'N'.
     ld a,true    ;Get setup for receiving position data from host.
     ld (n_o),a       ; n := true.
     ld (n_lo_o),a     ; n_low := true.
     ld c,h'4e
     call host
     jp l1_a

19:
     ld a,(chr)
     cp h'47          ; chr = "G" ?
     jp nz,l1_a       ; Jump if chr <> "G"
112:
     ld de, st_g    ; Here if n is false and last character
     call   s_pr_l_o     ; received from host was "G"
     ld a,true
     ld (g_o),a         ; g := true.
     ld c,h'47
     call host
     jp l1_a

114:
     ld a,(n_lo_o)     ; Here if data from host is to be received.
     cp false
     jp z,l17

115:
     ld a,(chr)        ; Here if LSB is to be received.
     ld (n_v_lo_o),a
     ld a,false
     ld (n_lo_o),a     ; n_low := false.
     ld c,ack
     call host
     jp l1_a
;
117:
     ld a,false         ; n_high := false.
     ld (n_o),a         ; n_0 := false.
     ld a,(chr)
     ld (n_v_hi_o),a ;n_val_high := last character received from host.

;
118:
;Read the shaft angle encoder into SAE0.  Send "interrogate" pulse.
    ld a,b'00000011  ; Set "interrogate" of SAE2, bit 0 of 8255A-4.
```

```
    out (porta4),a      ; Bit 1 stays high for second channel.
    out (porta4),a    ;delay.  Toggle bit 0.  Wait for "Data Ready."
    out (porta4),a
    ld a,b'00000010 ;Reset "interrogate" of SAE2, bit 0 of 8255A-4.
    out (porta4),a
10:                     ; SAE:= initial shaft angle reading.
    in a,(porta2)   ; Test "Data Ready" from sae2, bit 0 of 8255A-2.
    and b'00000001
;   jp  z,10            ; Loop until high.** restore for uS16/23**
    in a,(portb2)       ; Get least significant byte (from portb2).
    and b'11111000      ; *** remove this for uS16/23 operation ****
    ld (sae0_o),a         ;
    in a,(portc2)         ;Get most significant byte (from portc2).
    ld (sae0_o+h'01),a    ; sae0 (16 bits) is present sae reading. ;
;
;  Multiply sae reading by 1011/1024. 1011 = 001111110011B.
;  Reverse sae reading.
    ld hl,(sae0_o);       hl        := sae_o.
    ld a,l
     db h'2f           ; Complement accumulator, one's complement.
    ld l,a
    ld a,h
     db h'2f           ; One's complement, high byte.
    ld h,a
;
    ld (cc00),hl;         cc00      := - sae_o, used as buffer.
    ld (cc01),hl;         cc01      := - sae_o.
; >>>>>>
    add hl,hl         ;   [cy] < [h15  <  h0] < 0.
    sbc hl,hl
;
    ld (cc00+h'02),hl;    cc00+h'02 :=     h'ff or h'00.
    ld (cc01+h'02),hl;    cc01+h'02 :=     h'ff or h'00.
;
;                   cc00 will be destroyed in 'elements' routine.
    call elements; Generate cc02, cc04, cc08, cc16, cc32, cc64, cc128,
cc256 and cc512.
                  ; Above elements are generated from cc00.
;
;                           01 + 02.   Summing in cc00 (32 bit).
    ld hl,(cc01)              ;
    ld de,(cc02)
    add hl,de
    ld (cc00),hl
    ld hl,(cc01+h'02)
    ld de,(cc02+h'02)
    adc hl,de                  ;
    ld (cc00+h'02),hl          ;
                               ;
;                           16
    ld hl,(cc00)              ;
    ld de,(cc16)
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc16+h'02)
    adc hl,de                  ;
    ld (cc00+h'02),hl          ;
```

```
                                ;
;                          32
    ld hl,(cc00)                ;
    ld de,(cc32)
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc32+h'02)
    adc hl,de                   ;
    ld (cc00+h'02),hl           ;
                                ;
;                          64
    ld hl,(cc00)                ;
    ld de,(cc64)
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc64+h'02)
    adc hl,de                   ;
    ld (cc00+h'02),hl           ;
                                ;
;                         128
    ld hl,(cc00)                ;
    ld de,(cc128)
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc128+h'02)
    adc hl,de                   ;
    ld (cc00+h'02),hl           ;
                                ;
;                         256
    ld hl,(cc00)                ;
    ld de,(cc256)
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc256+h'02)
    adc hl,de                   ;
    ld (cc00+h'02),hl           ;
                                ;
;                         512
    ld hl,(cc00)                ;
    ld de,(cc512)
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc512+h'02)
    adc hl,de                   ; cc00 := cc512 + cc256 + cc128 +
    ld (cc00+h'02),hl           ;          cc64 +  cc32 +  cc16 +
                           ;          cc02 +  cc01 = 1011*cc00. ;
; Divide cc00 (32 bits) by 1024.
;
      call sr32                 ; cc00, hl := cc00 / 2.
      call sr32                 ; cc00, hl := cc00 / 2.
;
      ld a,(cc00+h'02)          ; Byte shift left.
      ld hl,(cc00)
```

```
      ld l,h
      ld h,a
      ld (sae0_o),hl  ; sae0_o := -1101/1024 of initial value.
;
;     < < < < < < < < < < < < < < < < < < < < < < < < < < < <
    ld a,(n_v_lo_o) ;sae1 from host data, n_val_low, n_val_high.
    ld (sae1_o),a
    ld a,(n_v_hi_o)
    ld (sae1_o+h'01),a   ; hl := sae1, destination angle.
;
118aa:
    ld a,(sae0_o+h'01)   ; high order byte.
    and b'10000000      ; test hight order bit.
    jp z,118b           ; jp if sae0(16) >= 0.

118c:
    ld a,(sae1_o+h'01) ; high order byte.    Case where sae0 < 0.
    and b'10000000
    jp z,118f           ; jp if sae1(16) >= 0.
118e:         ; Case where (sae1<0, sae0<0) or (sae1>=0, sae0>=0).
    ld hl,(sae0_o)        ; hl := sae0 (present position).
    ld de,(sae1_o)        ; de := sae1 (desired position).
    xor a                 ; cy := 0.
    sbc hl,de             ; hl := sae0 - sae1.
    ld a,h
    and b'10000000
    jp z,118d
118f:
    ld hl,(sae1_o)    ; hl := sae1. Case where sae1 >= 0, sae0 < 0.
    ld de,(sae0_o)        ; de := sae0.
    xor a                 ; cy := 0.
    sbc hl,de             ; hl := sae1 - sae0.
    ld c,h'2d             ; c := "-"                          *******
jp 118a
;
118b:                     ; Case where sae0 >= 0.
    ld a,(sae1_o+h'01)
    and b'10000000
    jp z,118e           ; Jump if sae1 >= 0.
;
118d:                     ; Case where sae0 >= 0, sae1 < 0.
    ld hl,(sae0_o)      ; hl := sae0.
    ld de,(sae1_o)      ; de := sae1.
    xor a               ; cy := 0.
    sbc hl,de           ; hl := sae0 - sae1.
    ld c,h'2b            ; c := "+".                       ********

118a:
    ld (cc),hl        ; save hl.
    call s_ch_1_o   ; Send direction to CY525-1, either "+" or "-".
    ld c,h'00
    call s_ch_1_o       ; Terminate direction command.
;
;   Need to multiply (cc) by 6080 to generate number of steps.
;   Double this factor for half-step operation.  February 23, 1990.
    ld hl,(cc);
    ld (cc00),hl    ;      cc00       := cc.
    ld hl,h'00
```

```
    ld (cc00+h'02),hl   ; cc00+h'02 := 00.
;
    ld hl,(cc00);
    add hl,hl
    ld (cc00),hl
    ld hl,(cc00+h'02);
    adc hl,hl
    ld (cc00+h'02),hl
;
    ld hl,(cc00);
    add hl,hl
    ld (cc00),hl
    ld hl,(cc00+h'02);
    adc hl,hl
    ld (cc00+h'02),hl
;
    ld hl,(cc00);
    add hl,hl
    ld (cc00),hl
    ld hl,(cc00+h'02);
    adc hl,hl
    ld (cc00+h'02),hl
;
    call elements; generate cc02, cc04, cc08, cc16, cc32, cc64, cc128,
cc256, cc512.
;                   Begin summation into cc00.
;                          08, 16
    ld hl,(cc08)              ;
    ld de,(cc16)
    add hl,de
    ld (cc00),hl

    ld hl,(cc08+h'02)
    ld de,(cc16+h'02)
    adc hl,de
    ld (cc00+h'02),hl          ; cc00 := cc08 + cc16.
;                          32
    ld hl,(cc00)              ;
    ld de,(cc32)
    add hl,de
    ld (cc00),hl

    ld hl,(cc00+h'02)
    ld de,(cc32+h'02)
    adc hl,de
    ld (cc00+h'02),hl          ; cc00 := cc08 + cc16 + cc32.
;                          64
    ld hl,(cc00)              ;
    ld de,(cc64)
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc64+h'02)
    adc hl,de
    ld (cc00+h'02),hl          ; cc00 := cc08 + cc16 + cc32 + cc64.
;                          128
    ld hl,(cc00)              ;
    ld de,(cc128)
```

```
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc128+h'02)
    adc hl,de
    ld (cc00+h'02),hl   ; cc00 := cc08 + cc16 + cc32 + cc64 + cc128.
;                         512
    ld hl,(cc00)              ;
    ld de,(cc512)
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc512+h'02)
    adc hl,de                  ; cc00 := cc08 + cc16 + cc32 + cc64 +
cc128
    ld (cc00+h'02),hl          ;       + cc512.

; February 23, 1990 - double value for half-step.

    ld hl,(cc00)
    add hl,hl
    ld (cc00),hl
    ld hl,(cc00+h'02)
    adc hl,hl
    ld (cc00+h'02),hl     ; February 23, 1990 - end.

;   <<<<<<<<<
;
    ld c,h'4e           ; Send the "N" command.
    call s_ch_1_o
    ld c,h'02
    call s_ch_1_o  ; Send byte count, 2, to CY525 for "N" command.
    ld a,(cc00+h'02)
    ld c,a
    call s_ch_1_o       ; No echo to host.
    ld a,(cc00+h'03)
    ld c,a
    call s_ch_1_o
    call host           ; Echo most significant byte to host.
    jp l1_a
;
; %%%%%%%%%%%%%        start of "p" loop          %%%%%%%%%%%%%%%
;
l1_p:
      ld a,dest_o
      ld (destination),a
      ld a,(g_p)
      cp false
      jp z,l5_p

l2_p:
      in   a,   (portb3)      ; "motion_complete" active high.
      and  b'00000001
      jp   nz,   l5_p
 l4_p:

        ld a,false          ;Set g_p = false.
        ld (g_p),a
```

```
15_p:
       ld a,(chr_ready)
       cp false
       jp z,11_a
;
       ld a,(channel)
       cp channelp
       jp nz,11_b      ; See if other channels needs the character.
16_p:
       ld a,(n_p)
       cp true
       jp z,114_p

17_p:
     ld a,(chr)
     cp h'4e           ; chr = 'N' ?
     jp nz,19_p

18_p:                    ; last character received was 'N'
     ld a,true    ; Get setup for receiving position data from host.
 ld (n_p),a        ; n := true.
     ld (n_lo_p),a     ; n_low := true.
     ld c,h'4e
     call host
     jp 11_a

19_p:
     ld a,(chr)
     cp h'47             ; Chr = "G" ?
     jp nz,11_a          ; Jump if chr <> "G"

112_p:                   ; Chr = 'G' here.
     ld de, st_g         ; Here if n is false and last character
     call   s_pr_3_p     ; received from host was "G"
     ld a,true
     ld (g_p),a         ; g := true.
     ld c,h'47
     call host
     jp 11_a

114_p:
     ld a,(n_lo_p)      ; Here if data from host is to be received.
     cp false
     jp z,117_p

115_p:
     ld a,(chr)        ; Here if LSB is to be received.
     ld (n_v_lo_p),a
     ld a,false
     ld (n_lo_p),a     ; n_low := false.
     ld c,ack
     call host
     jp 11_a
;
117_p:
     ld a,false         ; n_high := false.
     ld (n_p),a         ; n_0 := false.
```

```
     ld a,(chr)
     ld (n_v_hi_p),a ;n_val_high last character received from host.
;
118_p:
; Read the shaft angle encoder into SAE0.  Send "interrogate" pulse
    ld a,b'00000000   ; Set "interrogate" of SAE2, bit 1 of 8255A-4
    out (porta4),a     ; Active low, data freeze *.
    out (porta4),a     ; delay
    out (porta4),a
    in a,(portb4)      ; Get least significant byte.
    ld (sae0_p),a        ;
    in a,(portc4)
    ld (sae0_p+h'01),a   ; sae0 (16 bits) is present sae reading.
    ld a,b'00000010 ; Reset "interrogate" of SAE2, bit 1 of 8255A-4
    out (porta4),a
; Convert BCD into 2's compliment. 360 degrees=0111 1111 1111 1111b
    ld hl,d'00
    ld a,(sae0_p)
    ld c,a             ; Store least significant byte in c.
    and b'00000100
    jp z, z0002
    ld de, d'18        ; 0.1 degree.
    add hl,de
;
z0002:
    ld a,c
    and b'00001000
    jp z, z0004
    ld de,d'36         ; 0.2 degree.
    add hl,de
;
z0004:
    ld a,c
    and b'00010000
    jp z, z0008
    ld de,d'73
    add hl,de
;
z0008:
    ld a,c
    and b'00100000
    jp z, z0010
    ld de,d'146
    add hl,de
;
z0010:
    ld a,c
    and b'01000000
    jp z, z0020
    ld de,d'182        ; 1.0 degree.
    add hl,de
;
z0020:
    ld a,c
    and b'10000000
    jp z, z0040
    ld de,d'364
    add hl,de
```

```
;
z0040:
    ld a,(sae0_p+h'01)
    ld c,a
    and b'00000001
    jp z, z0080
    ld de,d'728
    add hl,de
;
z0080:
    ld a,c
    and b'00000010
    jp z, z0100
    ld de,d'1456
    add hl,de
;
z0100:
    ld a,c
    and b'00000100
    jp z, z0200
    ld de,d'1820        ; 10.0 degrees.
    add hl,de
;
z0200:
    ld a,c
    and b'00001000
    jp z, z0400
    ld de,d'3641
    add hl,de
;
z0400:
    ld a,c
    and b'00010000
    jp z, z0800
    ld de,d'7281
    add hl,de
;
z0800:
    ld a,c
    and b'00100000
    jp z, z1000
    ld de,d'14563
    add hl,de
;
z1000:
    ld a,c
    and b'01000000
    jp z, z2000
    ld de,d'18203       ; 100.0 degrees.
    add hl,de
;
z2000:
    ld a,c
    and b'10000000
    jp z, z4000
    ld de,d'36407
    add hl,de
;
```

```
z4000:
    ld a,l                      ; Restore (sae0_p) and
    (sae0_p+h'01).
    ld (sae0_p),a
    ld a,h
    ld (sae0_p+h'01),a
;
    ld a,(n_v_lo_p) ; sae1 from host data, n_val_low, n_val_high
    ld (sae1_p),a
    ld a,(n_v_hi_p)
    ld (sae1_p+h'01),a   ; hl := sae1, destination angle.
;
l18aa_p:
    ld a,(sae0_p+h'01)   ; high order byte.
    and b'10000000     ; test hight order bit.
    jp z,l18b_p          ; jp if sae0(16) >= 0.
;
l18c_p:
    ld a,(sae1_p+h'01) ; high order byte.
    and b'10000000
    jp z,l18f_p          ; jp if sae1(16) >= 0.
;
l18e_p:
    ld hl,(sae0_p)       ; hl := sae0.
    ld de,(sae1_p)       ; de := sae1.
    xor a              ; cy := 0.
    sbc hl,de          ; hl := sae0 - sae1.
    ld a,h
    and b'10000000
    jp z,l18d_p
;
l18f_p:
    ld hl,(sae1_p)       ; hl := sae1.
    ld de,(sae0_p)       ; de := sae0.
    xor a              ; cy := 0.
    sbc hl,de          ; hl := sae1 - sae0.
    ld c,h'2d          ; c := "-"                    *******
    jp l18a_p
;
l18b_p:
    ld a,(sae1_p+h'01)
    and b'10000000
    jp z,l18e_p
;
l18d_p:
    ld hl,(sae0_p)     ; hl := sae0.
    ld de,(sae1_p)     ; de := sae1.
    xor a              ; cy := 0.
    sbc hl,de          ; hl := sae0 - sae1.
    ld c,h'2b           ; c := "+".                  ********
;
l18a_p:
    ld (cc),hl       ; save hl.
    call s_ch_3_p    ; Send direction to CY525-1,  "+" or "-".
    ld c,h'00
    call s_ch_3_p      ; Terminate direction command.
;
    call mult200 ; cc00 cc01=200 cc (32 bit).  200 steps/turn.
```

```
    call elements; generate cc02, cc04, cc08, cc16, cc32, cc64,
    cc128, cc256 and cc512.
;                   Begin summation into cc00.
;                         512 + 256
    ld hl,(cc512)               ;
    ld de,(cc256)
    add hl,de
    ld (cc00),hl
    ld hl,(cc512+h'02)
    ld de,(cc256+h'02)
    adc hl,de
    ld (cc00+h'02),hl           ; cc00 := cc512 + cc256.
;                          32
    ld hl,(cc00)               ;
    ld de,(cc32)
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc32+h'02)
    adc hl,de
    ld (cc00+h'02),hl           ; cc00 := cc512 + cc256 + cc32.
    ld (cc00),hl
;
    ld c,h'4e           ; Send the "N" command.
    call s_ch_3_p
    ld c,h'02
    call s_ch_3_p   ; Send byte count, 2, to CY525 for "N" command.
    ld a,(cc00+h'02)
    ld c,a
    call s_ch_3_p
    ld a,(cc00+h'03)
    ld c,a
    call s_ch_3_p
    call host
    jp ll_a
; %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ;
; * * * * * * * * * *     P r o c e d u r e s   * * * * * * * * * *
;
;test serial i/o if input data ready.
;returns with a = true if input data ready (host), data byte in c.
;returns with a = false if input data from host not available.
;
test_s_in:
    in a,(portb5);
    and b'00000010
    jp z,zzz01
    in a,(porta5)
    ld (chr),a
    ld a,true
    ld (chr_ready),a
    ret
;
zzz01:
    ld a,false
    ld (chr_ready),a
    ret
;
;test serial i/o for data output ready (to host).
```

```
;
    ret

;
s_pr_3_p:          ; routine to send command bytes to the CY525, III.
                   ; de := pointer to byte string, 0ffh is stopper.
    ld a,(de)      ; get next byte from buffer.
    cp h'ff        ; is it the stopper?
    ret z
    inc de         ; update pointer.
    ld c,a
    call s_ch_3_p
    jp s_pr_3_p
;
s_ch_3_p:          ; output char in c to CY525.
    in a,(portb3)
    and ready
    jp z,s_ch_3_p
;
    ld a,c
    out (porta3),a   ; put character on the data bus.
;
    ld a,ioreq
    out (portd3),a ;reset (I/O request)*, tell CY525 data available ;
way3_p:
    in a,(portb3)
    and ready
    jp nz,way3_p
;
    ld a,noioreq     ; set bit 0 of port c to 1.
    out (portd3),a   ; set (I/O request)*.
    ret
;
mult200:           ; cc00 (32) := 200 * cc, (32 bit).
    ld hl,h'00     ;
    ld (cc+h'02),hl  ; Set upper word of cc to zero.
;
    ld hl,(cc)     ; cc := cc + cc;
    add hl,hl
    ld (cc),hl
    ld hl,(cc+h'02)
    adc hl,hl
    ld (cc+h'02),hl  ; cc:=2 phi.
;
    ld hl,(cc)
    add hl,hl
    ld (cc),hl
    ld (cc04),hl
    ld hl,(cc+h'02)
    adc hl,hl
    ld (cc+h'02),hl  ; cc:=4 phi.
    ld (cc04+h'02),hl  ; cc04 := 4 phi.
;
    ld hl,(cc)     ; cc := cc + cc;
    add hl,hl
    ld (cc),hl
    ld hl,(cc+h'02)
    adc hl,hl
```

```
    ld (cc+h'02),hl  ; cc:=8 phi.
;
    ld hl,(cc)       ; cc := cc + cc;
    add hl,hl
    ld (cc),hl
    ld hl,(cc+h'02)
    adc hl,hl
    ld (cc+h'02),hl  ; cc:=16 phi.
;
    ld hl,(cc)       ; cc := cc + cc;
    add hl,hl
    ld (cc),hl
    ld (cc32),hl
    ld hl,(cc+h'02)
    adc hl,hl
    ld (cc  +h'02),hl  ; cc   := 32 phi.
    ld (cc32+h'02),hl  ; cc32 := 32 phi.
;
    ld hl,(cc)       ; cc64 := cc + cc;
    add hl,hl
    ld (cc64),hl
    ld hl,(cc+h'02)
    adc hl,hl
    ld (cc64+h'02),hl  ; cc64 := 64 phi.
;                      Perform summation.
    ld hl,(cc64)
    ld de,(cc04)
    add hl,de
    ld (cc00),hl
    ld hl,(cc64+h'02)
    ld de,(cc04+h'02)
    adc hl,de
    ld (cc00+h'02),hl          ; cc00 := cc64 + cc04.
;
    ld hl,(cc00)               ;
    ld de,(cc32)
    add hl,de
    ld (cc00),hl
    ld hl,(cc00+h'02)
    ld de,(cc32+h'02)
    adc hl,de
    ld (cc00+h'02),hl           ; cc00 := cc64 + cc04 + cc32.
;
    ld hl,(cc00)                ; Double cc00.
    add hl,hl
    ld (cc00),hl
    ld (cc01),hl
    ld hl,(cc00+h'02)
    adc hl,hl
    ld (cc00+h'02),hl
    ld (cc01+h'02),hl           ; cc01 := cc00.
    ret
;
elements:           ; generate cc02, cc04, cc08, cc16, cc32, cc64,
                    ; cc128, cc256 and cc512, from cc00.
;                     02
    ld hl,(cc00)     ; cc00 := cc00 + cc00;
    add hl,hl
```

```
    ld (cc00),hl
    ld (cc02),hl
    ld hl,(cc00+h'02)
    adc hl,hl
    ld (cc00+h'02),hl  ; cc00:= 2 phi.
    ld (cc02+h'02),hl
;                        04
    ld hl,(cc00)       ; cc00 := cc00 + cc00;
    add hl,hl
    ld (cc00),hl
    ld (cc04),hl
    ld hl,(cc00+h'02)
    adc hl,hl
    ld (cc00+h'02),hl  ; cc00 := 4 phi.
    ld (cc04+h'02),hl  ; cc04 := 4 phi.
;                        08
    ld hl,(cc00)       ; cc00 := cc00 + cc00;
    add hl,hl
    ld (cc00),hl
    ld (cc08),hl
    ld hl,(cc00+h'02)
    adc hl,hl
    ld (cc00+h'02),hl  ; cc00 := 8 phi.
    ld (cc08+h'02),hl  ; cc08 := 8 phi.
;                        16
    ld hl,(cc00)       ; cc00 := cc00 + cc00;
    add hl,hl
    ld (cc00),hl
    ld (cc16),hl
    ld hl,(cc00+h'02)
    adc hl,hl
    ld (cc00+h'02),hl  ; cc00 := 16 phi.
    ld (cc16+h'02),hl  ; cc16 := 16 phi.
;                        32
    ld hl,(cc00)       ; cc00 := cc00 + cc00;
    add hl,hl
    ld (cc00),hl
    ld (cc32),hl
    ld hl,(cc00+h'02)
    adc hl,hl
    ld (cc00+h'02),hl  ; cc00 := 32 phi.
    ld (cc32+h'02),hl  ; cc32 := 32 phi.
;                        64
    ld hl,(cc00)       ; cc00 := cc00 + cc00;
    add hl,hl
    ld (cc00),hl
    ld (cc64),hl
    ld hl,(cc00+h'02)
    adc hl,hl
    ld (cc00+h'02),hl  ; cc00 := 64 phi.
    ld (cc64+h'02),hl  ; cc64 := 64 phi.
;                       128
    ld hl,(cc00)       ; cc00 := cc00 + cc00;
    add hl,hl
    ld (cc00 ),hl
    ld (cc128),hl
    ld hl,(cc00+h'02)
    adc hl,hl
```

```
    ld (cc00 +h'02),hl  ; cc00  := 128 phi.
    ld (cc128+h'02),hl  ; cc128 := 128 phi.
;                   256
    ld hl,(cc00)        ; cc00 := cc00 + cc00;
    add hl,hl
    ld (cc00),hl
    ld (cc256),hl
    ld hl,(cc00+h'02)
    adc hl,hl
    ld (cc00+h'02),hl
    ld (cc256+h'02),hl  ; cc256  := 256 phi.
;                   512
    ld hl,(cc00)        ; cc00 := cc00 + cc00;
    add hl,hl
    ld (cc00),hl
    ld (cc512),hl
    ld hl,(cc00+h'02)
    adc hl,hl
    ld (cc00+h'02),hl
    ld (cc512+h'02),hl  ; cc512  := 512 phi.
    ret
;
sr32:ld hl,(cc00+h'02)          ; 32 bit shift right from cc00
      db h'7c, h'1f, h'67       ; [ ?] > [h7 ... > ... h0] > [cy]
      db h'7d, h'1f, h'6f       ; [cy] > [l7 ... > ... l0] > [cy]
     ld (cc00+h'02),hl
;
     ld hl,(cc00)
      db h'7c, h'1f, h'67
      db h'7d, h'1f, h'6f
     ld (cc00),hl
     ret
;         < < < < < < < < < < < < < < < < < < < <
;
st_in_o:  db h'49, h'00           ;  "I"  ,initialize CY525_o.
          db h'46, h'01,   3      ;  "F," ,first_rate, .
          db h'52, h'01,  32      ;  "R"  ,rate,       .
          db h'53, h'01,   2      ;  "S"  ,slope,      .
          db h'5a, h'01,   2      ;  "Z"  ,Divisor     .
          db h'ff                 ;  "stopper."
;
st_in_p:  db h'49, h'00           ;  "I"  ,initialize CY525_p.
          db h'46, h'01,   3      ;  "F," ,first_rate, .
          db h'52, h'01,  32      ;  "R"  ,rate,       .
          db h'53, h'01,   2      ;  "S"  ,slope,      .
          db h'5a, h'01,   2      ;  "Z"  ,Divisor     .
          db h'ff                 ;  "stopper."
;
st_g:    db h'47, h'00, h'ff      ;  CY525: G, 0, "stopper."
st_on:   db h'42, h'00, h'ff      ;  Bit_set, CY525, pin 34.
st_off:  db h'43, h'00, h'ff      ;  Bit_reset, CY525, pin 34.
;
             ; initialize variables.
 equ true     ,h'00
 equ false    ,h'0ff
 equ ready    ,h'20
 equ ack      ,h'64 ; "d"
 equ ping     ,h'61 ; "a"
```

```
equ reset     ,h'62 ; "b"
equ ioselout  ,h'03
equ ioreq     ,h'00
equ noioreq   ,h'01
equ ioselin   ,h'02
equ porta1    ,h'08
equ portb1    ,h'09
equ portc1    ,h'0a
equ portd1    ,h'0b
equ porta2    ,h'00
equ portb2    ,h'01
equ portc2    ,h'02
equ portd2    ,h'03
equ porta3    ,h'0c
equ portb3    ,h'0d
equ portc3    ,h'0e
equ portd3    ,h'0f
equ porta4    ,h'04
equ portb4    ,h'05
equ portc4    ,h'06
equ portd4    ,h'07
equ porta5    ,h'10
equ portb5    ,h'11
equ channelo  ,h'4f ; "O"
equ channelp  ,h'50 ; "P"
equ g_status  ,h'53 ; "S"
equ dest_o    ,h'6f ; "o"
equ dest_p    ,h'70 ; "p"
;
;
end
```

INTENTIONALLY LEFT BLANK.

APPENDIX C:

PASCAL LISTING OF THE DRIVER PROGRAM UTILIZED BY THE HOST COMPUTER.

Note: Assembly Language "Hooks" Are Used to Allow the Program to Communicate With the Serial Input/Output Device for the Purpose of Controlling the Mount Computer.

INTENTIONALLY LEFT BLANK.

```
{File:CY525.pas, Disk#005, TurboDOS 1.2, March 14, 1989, June 26,1989}

{For use with stepper motor controllers: azimuth and elevation}
program CY525;

label s60;

var
 sae, i, max                                                    :integer;
 remote, g_o, g_p                                               :boolean;
 ping, ack, reset, rfda, status, data_lo, data_hi, cha          :byte;
 ch                                                             :char;
 g, n, o, p, s, channel                                         :byte;
 saer                                                           :real;

function inta(x:real):real;
         begin
            if x<0.0 then inta:=int(x)-1.0 else inta:=int(x);
         end;

function fract(x:real):real;
         begin
              if x<0.0 then fract:=frac(x)+1.0 else fract:=frac(x);
         end;

procedure inpmdm(var y,z:byte);
          begin
               inline($db/$05/$e6/$01/$2a/  y/$77);
               if y<> 0 then
               inline($db/$00/$2a/  z/$77);
          end;

procedure outmdm(var z:byte);
          begin
               inline($db/$05/$e6/$20/$28/$fa/$2a/  z/$7e/$d3/$00);
          end;

procedure mdmofl; {Procedure to take the ACE off line.}
          begin
               inline($af/$d3/$01/$db/$04/$f6/$10/$d3/$04);
          end;

procedure mdmonl; {Procedure to put the ACE on line.}
          begin
               inline($db/$00/$3e/$07);

inline($0e/$0c/$06/$46/$05/$20/$fd/$0d/$20/$f8/$3d/$20/$f3);
               inline($db/$00/$db/$04/$e6/$ef/$d3/$04);
          end;

procedure start_up;
          begin
               inline($21/$68/$00/$eb/$db/$03/$f6/$80/$d3/$03);
               inline($7b/$d3/$00/$7a);
               inline($3e/$03/$d3/$03/$db/$04/$f6/$03/$d3/$04);
          end;

procedure delay(z:integer);
```

```
          var i:integer;
          begin
               for i:=1 to z do begin end;
          end;

procedure convert(var x:integer; var y,z:byte);
          begin
               inline($2a/  x/$7e/$23/$5f/$7e/$2a/  z/$77/$2a/
y/$73);
          end;

begin {main}
      ping  := 97; {61h, "a"}
      reset := 98; {62h, "b"}
      ack   :=100; {64h, "d"}
      g     := 71; {47h, "G"}
      n     := 78; {4eh, "N"}
      o     := 79; {4fh, "O"}
      p     := 80; {50h, "P"}
      s     := 83; {53h, "S"}
      g_o   :=false;
      g_p   :=false;
      remote:=false; {****}

      clrscr;    start_up;

      cha:=0;
      while (cha<>ack) do
        begin
          status:= 0;
          while status = 0 do
            begin
              outmdm(ping);
              delay(9000);
              inpmdm(status,cha);
              if status = 0 then
                begin
                  gotoxy(1,1);
                  writeln('Remote not responding. ');
                  writeln('Please check power and prese the "reset"
switch.');
                  remote:=false;
                end; {if status}
            end; {while status}
        end; {while (cha<>ack)}


     if (status<>0) and (cha=ack) then remote:=true;

     if (remote=true) then writeln('Remote system has responded to a
"ping" ');

     while (remote = true) do
       begin

s60:     gotoxy(1,1);
         write(^['X']; {Enable cursor}
         write('Enter channel specifier, "o" or "p" : '); readln(ch);
```

```
          if (ch='o') then channel := o
        else if (ch='p') then channel := p
        else goto s60;

        delay(20000);   clrscr;

{Test status of channel}
        cha:=s; {Status}
        outmdm(cha); i:=1; status := 0;

        while ((status=0) and (i<400))do
          begin
            inpmdm(status,cha);
            i := i+1;
          end;

       if((status=0) and (i = 400)) then
       writeln(' System not responding.  "Status" ');
       remote:=false;
       if (status<>0) then remote:=true;


       if((cha=48) or (cha=50)) then g_o:=false;
       if((cha=48) or (cha=49)) then g_p:=false;

       if ( ( (channel=o) and (g_o=true) ) or
            ( (channel=p) and (g_p=true) ) ) then
         begin
           writeln('The channel you have selected is busy now.  ');
           goto s60;
         end;

       outmdm(channel); i := 1 ; status :=0;
       while ((status=0) and (i<400)) do
         begin
           inpmdm(status,cha);
           if((status<>0) and (cha<>ack)) then
             writeln('Not proper system response.  "Channel " ');
           i := i + 1;
         end;

       remote:=false;
       if ( (status<>0) and (cha=ack) ) then remote:= true;

       write('Enter shaft angle  ');
       readln(saer);
       sae := trunc(saer*32767.0/180.0);
       convert (sae,data_lo,data_hi);
       writeln;

{N}

       cha:=n;
       outmdm(cha); i:=1; status := 0;
       while ((status=0) and (i<400))do
         begin
          inpmdm(status,cha);
          if((status<>0) and (cha<>n)) then
```

```
            writeln('Not proper system response. ');
          i:=i+1;
        end;
      if ((status=0) and (i=400)) then
        writeln('System not responding. "N" ');

{data_lo}

      cha:=data_lo;
      outmdm(cha); i:=1; status := 0;
      while ( (status=0) and (i<400) ) do
        begin
          inpmdm(status,cha);
          if((status<>0) and (cha<>n)) then
            writeln('Not proper system response. ');
          i:=i+1;
        end;
      if ((status=0) and (i=400)) then
        writeln('System not responding. "Lo" ');


{Hi}

      cha:=data_hi;
      outmdm(cha); i:=1; status := 0;
      while ((status=0) and (i<400))do
        begin
          inpmdm(status,cha);
          if((status<>0) and (cha<>n)) then
          writeln('Not proper system response. ');
          i:=i+1;
        end;
      if ((status=0) and (i=400)) then
        begin
          writeln('System not responding. "Hi" ');
          writeln('The shaft angle encoder interface may be
defective ');
        end;

{The "cha" value is the hi byte of the "N" number send to the CY525.}
      writeln(cha:8);
      delay(9000);


{G}

      cha:=g;{G}
      max:=400;
      if (channel=o) then g_o:=true;
      if (channel=p) then g_p:=true;
      outmdm(cha); i:=1; status :=0;
      while((status=0) and (i<max)) do
        begin
          inpmdm(status,cha);
          if((status<>0) and (cha<>g)) then
            writeln('Not proper system response.');
          i:=i+1;
        end;
```

```
        if ((status=0) and (i=max)) then
          writeln('System not responding.  "G" ');
        remote:=false;
        if((status<>0) and (cha=g)) then remote:=true;
      end;
end.
^Z
```

INTENTIONALLY LEFT BLANK.

No. of Copies	Organization
2	Administrator Defense Technical Info Center ATTN: DTIC-DDA Cameron Station Alexandria, VA 22304-6145
1	Commander U.S. Army Materiel Command ATTN: AMCAM 5001 Eisenhower Ave. Alexandria, VA 22333-0001
1	Commander U.S. Army Laboratory Command ATTN: AMSLC-DL 2800 Powder Mill Rd. Adelphi, MD 20783-1145
2	Commander U.S. Army Armament Research, Development, and Engineering Center ATTN: SMCAR-IMI-I Picatinny Arsenal, NJ 07806-5000
2	Commander U.S. Army Armament Research, Development, and Engineering Center ATTN: SMCAR-TDC Picatinny Arsenal, NJ 07806-5000
1	Director Benet Weapons Laboratory U.S. Army Armament Research, Development, and Engineering Center ATTN: SMCAR-CCB-TL Watervliet, NY 12189-4050
(Unclass. only)1	Commander U.S. Army Rock Island Arsenal ATTN: SMCRI-TL/Technical Library Rock Island, IL 61299-5000
1	Director U.S. Army Aviation Research and Technology Activity ATTN: SAVRT-R (Library) M/S 219-3 Ames Research Center Moffett Field, CA 94035-1000
1	Commander U.S. Army Missile Command ATTN: AMSMI-RD-CS-R (DOC) Redstone Arsenal, AL 35898-5010

No. of Copies	Organization
1	Commander U.S. Army Tank-Automotive Command ATTN: ASQNC-TAC-DIT (Technical Information Center) Warren, MI 48397-5000
1	Director U.S. Army TRADOC Analysis Command ATTN: ATRC-WSR White Sands Missile Range, NM 88002-5502
1	Commandant U.S. Army Field Artillery School ATTN: ATSF-CSI Ft. Sill, OK 73503-5000
(Class. only)1	Commandant U.S. Army Infantry School ATTN: ATSH-CD (Security Mgr.) Fort Benning, GA 31905-5660
(Unclass. only)1	Commandant U.S. Army Infantry School ATTN: ATSH-CD-CSO-OR Fort Benning, GA 31905-5660
1	WL/MNOI Eglin AFB, FL 32542-5000
	Aberdeen Proving Ground
2	Dir, USAMSAA ATTN: AMXSY-D AMXSY-MP, H. Cohen
1	Cdr, USATECOM ATTN: AMSTE-TC
3	Cdr, CRDEC, AMCCOM ATTN: SMCCR-RSP-A SMCCR-MU SMCCR-MSI
1	Dir, VLAMO ATTN: AMSLC-VL-D
10	Dir, USABRL ATTN: SLCBR-DD-T

INTENTIONALLY LEFT BLANK.

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. BRL Report Number BRL-TR-3382 Date of Report July 1992

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

Name

Organization

Address

City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the New or Correct Address in Block 6 above and the Old or Incorrect address below.

OLD ADDRESS

Name

Organization

Address

City, State, Zip Code

(Remove this sheet, fold as indicated, staple or tape closed, and mail.)

DEPARTMENT OF THE ARMY
Director
U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-5066

OFFICIAL BUSINESS

BUSINESS REPLY MAIL
FIRST CLASS PERMIT No 0001, APG, MD

Postage will be paid by addressee.

Director
U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-5066

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES